**Part 1 (A): Spring Boot Core — Batch 1 (Q1–Q12)**

**Q1. What does @SpringBootApplication do?**

**Answer:**  
It’s a composite annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. This makes it the main entry point for Spring Boot applications.  
When you run your app via SpringApplication.run(), Boot automatically scans for components and triggers auto-configuration.

**Code Example:**

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**💡 Interview Insight:**  
Interviewers expect you to mention that @EnableAutoConfiguration drives Boot’s “magic.” Bonus points if you mention that Boot looks at spring.factories in the JARs to determine configurations.

**Q2. What is Auto-Configuration in Spring Boot?**

**Answer:**  
Auto-configuration automatically configures your application based on the dependencies on your classpath.  
It uses @Conditional annotations (like @ConditionalOnClass, @ConditionalOnMissingBean) to decide which beans to load.

**Code Example:**

@Configuration

@ConditionalOnClass(DataSource.class)

public class DataSourceAutoConfiguration {

// Automatically configures a DataSource bean if on classpath

}

**💡 Interview Insight:**  
Mention that auto-configuration is the key to Boot’s convention-over-configuration philosophy. Senior interviewers often ask how to disable it — @SpringBootApplication(exclude = DataSourceAutoConfiguration.class).

**Q3. What are Spring Boot Starters?**

**Answer:**  
Starters are curated dependency descriptors that simplify Maven or Gradle configuration. Each starter brings in a set of dependencies to build a particular type of application.

**Code Example (Maven):**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

**💡 Interview Insight:**  
Explain that starters are opinionated, maintained by the Spring team, and reduce version conflicts.

**Q4. What is the SpringApplication class responsible for?**

**Answer:**  
SpringApplication bootstraps the Spring context. It:

1. Sets up the environment.
2. Creates and refreshes the ApplicationContext.
3. Loads configuration properties.
4. Starts the embedded server (if web app).

**Code Example:**

public static void main(String[] args) {

SpringApplication.run(MyApplication.class, args);

}

**💡 Interview Insight:**  
Mention that you can customize startup behavior via SpringApplicationBuilder or listeners like ApplicationStartingEvent.

**Q5. How do you enable different profiles in Spring Boot?**

**Answer:**  
Use spring.profiles.active in properties or command-line arguments to specify which profile(s) to activate.

**Code Example:**

spring.profiles.active=dev

@Profile("dev")

@Component

public class DevDataSourceConfig {}

**💡 Interview Insight:**  
Mention that you can define multiple profiles (comma-separated) and use spring.profiles.include for layered configurations.

**Q6. What’s the difference between application.properties and application.yml?**

**Answer:**  
Both define configurations, but YAML allows a more hierarchical structure and cleaner grouping.

**Code Example:**

server:

port: 8081

spring:

datasource:

url: jdbc:mysql://localhost:3306/test

**💡 Interview Insight:**  
You’ll impress if you mention that .yml supports multiple documents using --- (useful for multiple profiles in one file).

**Q7. How does Spring Boot load externalized configurations?**

**Answer:**  
It follows a defined **property source order**, from highest to lowest precedence:

1. Command-line arguments
2. Environment variables
3. application.properties/yml
4. @PropertySource annotations
5. Default values in code

**💡 Interview Insight:**  
Senior interviewers expect you to mention EnvironmentPostProcessor for custom config loading (like pulling secrets from Vault).

**Q8. What’s the difference between @Component, @Service, @Repository, and @Controller?**

**Answer:**  
They all register beans, but with semantic meaning:

* @Component: Generic bean
* @Service: Business logic
* @Repository: Data access layer (adds exception translation)
* @Controller: MVC Controller

**💡 Interview Insight:**  
Mention that the distinction helps in AOP and stereotype scanning.

**Q9. How do you customize the banner in Spring Boot?**

**Answer:**  
Add a banner.txt or banner.png in src/main/resources.

**Code Example (banner.txt):**

Welcome to MyApp!

**💡 Interview Insight:**  
Mention spring.main.banner-mode=off disables it, and you can use placeholders like ${spring-boot.version} in banners.

**Q10. How can you run code on startup in Spring Boot?**

**Answer:**  
Implement CommandLineRunner or ApplicationRunner.

**Code Example:**

@Component

public class StartupRunner implements CommandLineRunner {

public void run(String... args) {

System.out.println("App started!");

}

}

**💡 Interview Insight:**  
Explain difference: ApplicationRunner gives you access to ApplicationArguments for structured argument handling.

**Q11. What is the role of @Configuration and @Bean?**

**Answer:**  
@Configuration marks a class for bean definitions; @Bean defines a bean inside it.

**Code Example:**

@Configuration

public class MyConfig {

@Bean

public MyService myService() {

return new MyServiceImpl();

}

}

**💡 Interview Insight:**  
Mention that @Configuration classes are subclassed using CGLIB to handle inter-bean dependencies correctly.

**Q12. What is the Spring Boot DevTools module used for?**

**Answer:**  
It provides developer conveniences like automatic restart, live reload, and property overrides during development.

**💡 Interview Insight:**  
Interviewers might ask how to disable restarts for large projects — mention spring.devtools.restart.enabled=false.

**🧩 Part 1 (A): Spring Boot Core — Batch 2 (Q13–Q25)**

**Q13. What are conditional beans in Spring Boot?**

**Answer:**  
Conditional beans allow a bean to be created only if certain conditions are met, like a property value, class presence, or missing bean.

**Code Example:**

@Bean

@ConditionalOnProperty(name = "feature.enabled", havingValue = "true")

public FeatureService featureService() {

return new FeatureServiceImpl();

}

**💡 Interview Insight:**  
Senior candidates should mention @ConditionalOnClass, @ConditionalOnMissingBean, and that these annotations help build modular, environment-specific configurations.

**Q14. What is the difference between @ConditionalOnProperty and @Profile?**

**Answer:**

* @ConditionalOnProperty checks a property value to conditionally register a bean.
* @Profile activates beans based on active Spring profiles.

**💡 Interview Insight:**  
They may ask how you’d combine them to create multi-environment features. Example: @Profile("dev") + @ConditionalOnProperty("featureX.enabled").

**Q15. How do you listen to Spring Boot application events?**

**Answer:**  
Use @EventListener or implement ApplicationListener<T>.

**Code Example:**

@Component

public class StartupListener {

@EventListener

public void handleContextRefreshed(ContextRefreshedEvent event) {

System.out.println("Context refreshed!");

}

}

**💡 Interview Insight:**  
Senior follow-up: can events be asynchronous? Answer: yes, add @Async and enable async support.

**Q16. What is the difference between ApplicationReadyEvent and ContextRefreshedEvent?**

**Answer:**

* ContextRefreshedEvent fires after ApplicationContext is initialized or refreshed.
* ApplicationReadyEvent fires after all beans are loaded and the application is ready to service requests.

**💡 Interview Insight:**  
Mention that ApplicationReadyEvent is the last event during startup, useful for warm-up tasks.

**Q17. What is Environment in Spring Boot?**

**Answer:**  
Environment abstracts property sources and profiles. You can access property values and check active profiles programmatically.

**Code Example:**

@Autowired

private Environment env;

public void logPort() {

System.out.println(env.getProperty("server.port"));

}

**💡 Interview Insight:**  
Advanced: explain MutablePropertySources and adding custom sources at runtime.

**Q18. How do you inject configuration properties using @Value vs @ConfigurationProperties?**

**Answer:**

* @Value injects single property values.
* @ConfigurationProperties binds multiple related properties into a POJO.

**Code Example:**

@Component

@ConfigurationProperties(prefix = "app")

public class AppConfig {

private String name;

private int timeout;

// getters and setters

}

**💡 Interview Insight:**  
Use @ConfigurationProperties for structured config and validation, @Value for simple injections.

**Q19. How do you manage the lifecycle of beans in Spring Boot?**

**Answer:**

* @PostConstruct / @PreDestroy
* Implement InitializingBean / DisposableBean
* Use SmartLifecycle for advanced startup/shutdown sequencing.

**💡 Interview Insight:**  
Mention differences between afterPropertiesSet() and @PostConstruct, and when you might need SmartLifecycle for ordered startup.

**Q20. How do you handle graceful shutdown in Spring Boot?**

**Answer:**  
Enable graceful shutdown in application.properties:

server.shutdown=graceful

spring.lifecycle.timeout-per-shutdown-phase=30s

Boot waits for active requests to finish before closing.

**💡 Interview Insight:**  
Tie this into Kubernetes SIGTERM handling for production deployments.

**Q21. How do you enable logging in Spring Boot?**

**Answer:**  
Spring Boot uses **Logback** by default. You can configure via application.properties or logback-spring.xml.

**Code Example:**

logging.level.org.springframework=INFO

logging.level.com.myapp=DEBUG

logging.file.name=app.log

**💡 Interview Insight:**  
Senior interviewers may ask how to configure different logging for dev vs prod or external logging appenders.

**Q22. What is the Spring Boot Actuator?**

**Answer:**  
Actuator provides production-ready endpoints for monitoring, metrics, and health checks.

**Code Example (Maven dependency):**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

**💡 Interview Insight:**  
Mention endpoints like /actuator/health, /metrics, and how to secure them.

**Q23. How can you add custom metrics to Actuator?**

**Answer:**  
Use Micrometer’s MeterRegistry to create counters, timers, and gauges.

**Code Example:**

@Component

public class OrderMetrics {

public OrderMetrics(MeterRegistry registry) {

Counter.builder("orders.created")

.description("Number of orders created")

.register(registry);

}

}

**💡 Interview Insight:**  
Show you know how to integrate with Prometheus, Datadog, or other monitoring systems.

**Q24. What is the difference between CommandLineRunner and ApplicationRunner?**

**Answer:**

* CommandLineRunner provides String[] args.
* ApplicationRunner provides ApplicationArguments (access to named options and non-option args).

**💡 Interview Insight:**  
Explain that ApplicationRunner is better for structured argument processing and is more flexible in production apps.

**Q25. How do you customize the Spring Boot startup sequence?**

**Answer:**  
You can customize using:

* SpringApplicationBuilder
* Custom initializers (ApplicationContextInitializer)
* Listeners (ApplicationListener)
* Banner mode and lazy initialization

**Code Example:**

SpringApplication app = new SpringApplication(MyApp.class);

app.setBannerMode(Banner.Mode.OFF);

app.setLazyInitialization(true);

app.addInitializers(new MyInitializer());

app.run(args);

**💡 Interview Insight:**  
Interviewers may ask how you control startup order in large applications with multiple contexts and modules.

**🧩 Part 1 (B): Spring Boot Core — Bonus Section**

**Senior-Level Behavioral / Situational Questions**

**Q26. Describe a time you optimized a Spring Boot microservice for performance.**  
**Answer:**

* Identify bottlenecks using profiling tools (VisualVM, JProfiler, Micrometer metrics).
* Optimize queries (avoid N+1 in JPA).
* Use caching (Spring Cache, Redis).
* Optimize thread pools for async processing.

**💡 Interview Insight:**  
Interviewer expects you to explain measurable improvements (latency reduction, throughput increase) and decisions behind caching, async tasks, or DB optimization.

**Q27. How have you handled a critical production issue in a Spring Boot application?**  
**Answer:**

* Quickly identify root cause using logs, metrics, and APM.
* Rollback deployment if necessary.
* Apply hotfix or feature flag to mitigate issue.
* Conduct post-mortem and implement preventive measures (monitoring, circuit breakers).

**💡 Interview Insight:**  
Emphasize problem-solving, ownership, and communication with stakeholders.

**Q28. Explain a design decision you made that improved the scalability of a Spring Boot system.**  
**Answer:**

* Introduced horizontal scaling with stateless services.
* Added load balancing with Spring Cloud / Ribbon.
* Applied caching layers for frequently accessed data.
* Optimized database access patterns with pagination and indexing.

**💡 Interview Insight:**  
Show that you can reason about scalability trade-offs: stateless vs stateful, sync vs async processing, and when to use queues or caching.

**Q29. How do you decide between synchronous and asynchronous processing in Spring Boot?**  
**Answer:**

* Use synchronous for quick, sequential tasks where immediate response is needed.
* Use @Async or message queues (Kafka, RabbitMQ) for long-running tasks.
* Consider system load, fault tolerance, and response time SLAs.

**💡 Interview Insight:**  
Highlight decision-making process — why async was chosen, how failures are handled, and how you monitor background tasks.

**Q30. Describe a time you refactored a Spring Boot service to improve maintainability.**  
**Answer:**

* Split a monolithic service into smaller, single-responsibility components.
* Extracted reusable configurations, services, and DTOs.
* Introduced proper exception handling, logging, and consistent naming conventions.

**💡 Interview Insight:**  
Demonstrates clean code practices, modularity, and leadership in improving existing systems.

**🧩 Part 2: Spring Data JPA & Hibernate — Batch 1 (Q51–Q75)**

**Q51. What is Spring Data JPA?**

**Answer:**  
Spring Data JPA simplifies data access by providing repository abstractions on top of JPA. It allows CRUD operations, query methods, and pagination without boilerplate code.

**Code Example:**

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByLastName(String lastName);

}

**💡 Interview Insight:**  
Senior candidates should mention how Spring Data JPA integrates with Hibernate and supports custom queries, pagination, and projections.

**Q52. What are the main interfaces in Spring Data JPA?**

**Answer:**

* CrudRepository — basic CRUD operations
* PagingAndSortingRepository — adds pagination and sorting
* JpaRepository — extends the above, adds JPA-specific methods

**💡 Interview Insight:**  
Explain why JpaRepository is often used for advanced enterprise applications.

**Q53. What is the difference between @Entity and @Table annotations?**

**Answer:**

* @Entity marks a class as a JPA entity.
* @Table specifies the table name and schema in the database.

**Code Example:**

@Entity

@Table(name = "users")

public class User {

@Id

private Long id;

private String firstName;

private String lastName;

}

**💡 Interview Insight:**  
Interviewers may ask about default naming strategies and schema mapping.

**Q54. What is the difference between @OneToOne, @OneToMany, @ManyToOne, and @ManyToMany?**

**Answer:**  
These annotations define entity relationships:

* @OneToOne: Single reference between two entities
* @OneToMany: One entity to multiple entities
* @ManyToOne: Multiple entities point to one entity
* @ManyToMany: Multiple entities relate to multiple entities

**💡 Interview Insight:**  
Be ready to explain **fetch types** (EAGER vs LAZY) and cascading options.

**Q55. How do you configure fetch types in JPA?**

**Answer:**

@OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)

private List<Order> orders;

**💡 Interview Insight:**  
Senior interviews often focus on N+1 query problems and how to optimize fetch strategies.

**Q56. How do you define a composite primary key in JPA?**

**Answer:**  
Use @Embeddable and @EmbeddedId.

**Code Example:**

@Embeddable

public class OrderId implements Serializable {

private Long orderNumber;

private Long productId;

}

@Entity

public class Order {

@EmbeddedId

private OrderId id;

}

**💡 Interview Insight:**  
Be ready to explain equals/hashCode implementations in embedded keys.

**Q57. How do you write custom queries in Spring Data JPA?**

**Answer:**

* Using @Query annotation with JPQL or native SQL
* Method naming convention (query derivation)

**Code Example:**

@Query("SELECT u FROM User u WHERE u.email = ?1")

User findByEmail(String email);

**💡 Interview Insight:**  
Mention parameter binding (?1 vs :email) and when to prefer native queries.

**Q58. What is the difference between save() and saveAndFlush()?**

**Answer:**

* save(): persists the entity but may defer flushing
* saveAndFlush(): persists and immediately flushes changes to the database

**💡 Interview Insight:**  
Good to mention implications on transaction boundaries and auto-commit behavior.

**Q59. How do you implement pagination and sorting?**

**Answer:**

Pageable pageable = PageRequest.of(0, 10, Sort.by("lastName").ascending());

Page<User> page = userRepository.findAll(pageable);

**💡 Interview Insight:**  
Senior interviews may ask about performance implications of large datasets and optimizing queries.

**Q60. How do you manage transactions in Spring Boot?**

**Answer:**  
Use @Transactional at class or method level.

**Code Example:**

@Service

@Transactional

public class UserService {

public void createUser(User user) { ... }

}

**💡 Interview Insight:**  
Explain propagation, isolation levels, and rollback behavior for advanced interviews.

**Q61. Difference between EntityManager and JpaRepository?**

**Answer:**

* JpaRepository provides high-level CRUD abstraction
* EntityManager provides low-level API for queries, batch operations, and fine-grained control

**💡 Interview Insight:**  
Senior developers should know when to use one over the other for performance tuning.

**Q62. What is the difference between @Transactional(readOnly=true) and default transaction?**

**Answer:**

* readOnly=true hints the persistence provider to optimize for read operations.
* Default is read-write, allows insert/update/delete.

**💡 Interview Insight:**  
Important for large read-heavy services; may reduce locking and flush behavior.

**Q63. How do you implement optimistic locking?**

**Answer:**  
Use @Version annotation.

**Code Example:**

@Version

private Long version;

**💡 Interview Insight:**  
Interviewers want to see awareness of concurrency issues and how optimistic locking prevents lost updates.

**Q64. How do you implement pessimistic locking?**

**Answer:**  
Use @Lock(LockModeType.PESSIMISTIC\_WRITE) on queries or EntityManager.lock().

**💡 Interview Insight:**  
Senior candidates should mention performance impact and deadlock considerations.

**Q65. How do you map enums in JPA?**

**Answer:**

@Enumerated(EnumType.STRING)

private Status status;

**💡 Interview Insight:**  
Explain difference between ORDINAL and STRING mapping — using STRING is safer for DB schema changes.

**Q66. How do you handle lazy loading exceptions?**

**Answer:**

* Fetch data within transactional context
* Use JOIN FETCH in queries
* Use @Transactional on service methods

**💡 Interview Insight:**  
Be ready to explain LazyInitializationException and ways to avoid it in real apps.

**Q67. How do you batch insert/update efficiently?**

**Answer:**

* Use saveAll() or EntityManager.persist() in batches
* Configure Hibernate batching:

spring.jpa.properties.hibernate.jdbc.batch\_size=50

**💡 Interview Insight:**  
Interviewers may ask for performance metrics and memory usage considerations.

**Q68. How do you map one-to-many relationships with join tables?**

**Answer:**

@ManyToMany

@JoinTable(

name = "user\_roles",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "role\_id")

)

private Set<Role> roles;

**💡 Interview Insight:**  
Highlight cascade, fetch type, and performance trade-offs.

**Q69. How do you handle database migrations in Spring Boot?**

**Answer:**  
Use Flyway or Liquibase for versioned schema migrations.

**Code Example:**

spring.flyway.enabled=true

spring.flyway.locations=classpath:db/migration

**💡 Interview Insight:**  
Explain rollback strategies, repeatable scripts, and CI/CD integration.

**Q70. How do you use projections in Spring Data JPA?**

**Answer:**  
Projections allow selecting partial entity data.

**Code Example:**

public interface UserNameOnly {

String getFirstName();

String getLastName();

}

List<UserNameOnly> findByLastName(String lastName);

**💡 Interview Insight:**  
Good for reducing payload size and improving query performance.

**Q71. What are native queries and when to use them?**

**Answer:**

* Native queries are raw SQL queries using @Query(nativeQuery = true)
* Useful for complex queries not easily expressed in JPQL

**💡 Interview Insight:**  
Explain trade-offs: DB portability vs performance optimization.

**Q72. How do you handle soft deletes in JPA?**

**Answer:**  
Add a deleted flag and filter queries using @Where (Hibernate) or global filters.

**Code Example:**

@Where(clause="deleted=false")

@Entity

public class User { ... }

**💡 Interview Insight:**  
Senior candidates should mention pros/cons: auditability vs query complexity.

**Q73. How do you integrate Spring Data JPA with caching?**

**Answer:**  
Use Spring Cache annotations (@Cacheable, @CacheEvict) on repository/service methods.

**Code Example:**

@Cacheable("users")

public User findById(Long id) { ... }

**💡 Interview Insight:**  
Highlight cache invalidation strategies and impact on transactional consistency.

**Q74. How do you monitor JPA performance in production?**

**Answer:**

* Enable SQL logging: spring.jpa.show-sql=true
* Use hibernate.format\_sql and hibernate.generate\_statistics
* Integrate APM tools (New Relic, AppDynamics)
* Monitor slow queries and N+1 problems

**💡 Interview Insight:**  
Senior-level: discuss metrics, dashboards, and alerting.

**Q75. How do you handle complex relationships in large enterprise schemas?**

**Answer:**

* Normalize entities and use DTOs for data transfer
* Avoid deep nested relationships in queries
* Use JOIN FETCH, batch fetching, or projections
* Consider read-only views for reporting

**💡 Interview Insight:**  
Demonstrates architectural thinking and real-world experience.

Top of Form

Bottom of Form

Top of Form

Bottom of Form